

Enumerating regular expressions and their languages

Hermann Gruber¹, Jonathan Lee², Jeffrey Shallit³

¹ Institut für Informatik, Justus-Liebig-Universität Giessen
Arndtstrasse 2
D-35392 Giessen, Germany

² Department of Mathematics, Stanford University
Building 380, Sloan Hall
Stanford, CA 94305, United States of America

³ School of Computer Science, University of Waterloo
Waterloo, ON N2L 3G1, Canada
email: hermann.gruber@informatik.uni-giessen.de,
jlee@math.stanford.edu,
shallit@cs.uwaterloo.ca

2010 Mathematics Subject Classification: 68Q45

Key words: Finite automata, regular expressions, combinatorial enumeration.

Contents

1	Introduction and overview	2
2	On measuring the size of a regular expression	3
3	A simple grammar for valid regular expressions	4
4	Unambiguous context-free grammars and the Chomsky-Schützenberger theorem	5
5	Solving algebraic equations using Gröbner bases	8
6	Asymptotic bounds via singularity analysis	10
7	Lower bounds on enumeration of regular languages by regular expressions	13
7.1	Trie representations for finite languages	14
7.2	Trie representations for some infinite regular languages	19
8	Upper bounds on enumeration of regular languages by regular expressions	22
8.1	A grammar based on normalized regular expressions	22
8.2	A grammar based on strong star normal form	24
9	Exact enumerations	26
10	Conclusion and open problems	27

1 Introduction and overview

Regular expressions have been studied for almost fifty years, yet many interesting and challenging problems about them remain unsolved. By a regular expression, we mean a string over the alphabet $\Sigma \cup \{+, *, (,), \varepsilon, \emptyset\}$ that represents a regular language. For example, $(0 + 10)^* (1 + \varepsilon)$ represents the language of all strings over $\{0, 1\}$ that do not contain two consecutive 1's.

We would like to enumerate both (i) valid regular expressions and (ii) the distinct languages they represent. Observe that these are two different enumeration tasks: on the one hand, every regular expression represents exactly one regular language. On the other hand, simple examples, such as the expressions $(a + b)^*$ and $(b * a^*)^*$, show that there is no one-to-one correspondence between regular languages and regular expressions.

We are in a similar situation if we use descriptors other than regular expressions, such as deterministic or nondeterministic finite automata. Although enumeration of automata has a long history, until recently little attention was paid to enumerating the distinct languages accepted. Instead authors concentrated on enumerating the automata themselves according to various criteria (e.g., acyclic, nonisomorphic, strongly connected, initially connected, ...).

Here is a brief survey of known results on automata. Vyssotsky [50] raised the question of enumerating strongly connected finite automata in an obscure technical report (but we have not been able to obtain a copy). Harary [16] enumerated the number of “functional digraphs” (which are essentially unary deterministic automata with no distinguished initial or final states) according to their cycle structure; also see Read [45] and [37]. Harary also mentioned the problem of enumerating deterministic finite automata over a binary alphabet as an open problem in a 1960 survey of open problems in enumeration [17, pp. 75,87], and later in a similar 1964 survey [18]. Ginsburg [13, p. 18] asked for the number of nonisomorphic automata with output on n states with given input and output alphabet size.

Harrison [20, 21] developed exact formulas for the number of automata with specified size of the input alphabet, output alphabet, and number of states. Similar results were found by Korshunov [27]. However, in their model, the automata do not have a distinguished initial state or set of final states. Using the same model, Radke [43] enumerated the number of strongly connected automata, but his solution was very complicated and not particularly useful. Harary and Palmer [19] found very complicated formulas in the same model, but including an initial state and any number of final states.

Harrison [20, 21] gave asymptotic estimates for the number of automata in his model, but his formulas contained some errors that were later corrected by Korshunov [28]. For example, the number of nonisomorphic unary automata with n states (and no distinguished initial or final states) is asymptotically $c(\pi n)^{-\frac{1}{2}} \tau^{-n}$ where $c \doteq 0.80$ and $\tau \doteq 0.34$.

Much work on enumeration of automata was done in the former Soviet Union. For example, Liskovets [35] studied the number of initially connected automata and gave both a recurrence formula and an asymptotic formula for them; also see Robinson [46]. Korshunov [29] counted the number of minimal automata, and [30] gave asymptotic estimates for the number of initially connected automata. The 78-page survey by Korshunov [31], which unfortunately seems to never have been translated into English, gives these

and many other results. More recently, Bassino and Nicaud [2] found that the number of nonisomorphic initially connected deterministic automata with n states is closely related to the Stirling numbers of the second kind.

Shallit and Breitbart observed that the number of finite automata can be applied to give bounds on the “automaticity” of languages and functions [48]. Pomerance, Robson, and Shallit [42] gave an upper bound on the number of distinct unary languages accepted by unary NFA’s with n states. Domaratzki, Kisman, and Shallit considered the number of distinct languages accepted by finite automata with n states [9]. They showed, for example, that the number of distinct languages accepted by unary finite automata with n states is $2^n(n - \alpha + O(n2^{-n/2}))$, where $\alpha \doteq 1.3827$. (A weaker result was previously obtained by Nicaud [40].) Domaratzki [6, 7] gave bounds on the number of minimal DFA’s accepting finite languages, which were improved by Liskovets [36]. Also see [3]. For more details about enumeration of automata and languages, see the survey of Domaratzki [8].

2 On measuring the size of a regular expression

Although, as we have seen, there has been much work for over 50 years on enumerating automata and the languages they represent, the analogous problem for regular expressions does not seem to have been studied before 2004 [33]. We define $R_k(n)$ to be the number of distinct languages specified by regular expressions of size n over a k -letter alphabet. The “size” of a regular expression can be defined in several different ways [11]:

- *Ordinary length*: total number of symbols, including parentheses, \emptyset , ε , etc., counted with multiplicity.
 - $(0 + 10) * (1 + \varepsilon)$ has ordinary length 12
 - Mentioned, for example, in [1, p. 396], [25].
- *Reverse polish length*: number of symbols in a reverse polish equivalent, including a symbol \bullet for concatenation. Equivalently, number of nodes in a syntax tree for the expression.
 - $(0 + 10) * (1 + \varepsilon)$ in reverse polish would be $010 \bullet + * \varepsilon + \bullet$
 - This has reverse polish length 10
 - Mentioned in [52]
- *Alphabetic width*: number of symbols from Σ , counted with multiplicity, not including ε , \emptyset , parentheses, operators
 - $(0 + 10) * (1 + \varepsilon)$ has alphabetic width 4
 - Mentioned in [39, 10, 34]

Each size measure seems to have its own advantages and disadvantages. The ordinary length appears to be the most direct way to measure the size of a regular expression. Here we can employ the usual priority rules, borrowed from arithmetic, for saving parentheses and omitting the \bullet operator. This favors the catenation operator \bullet over the union operator $+$. For instance, the expression $(a \bullet b) + (c \bullet d)$ can be written more briefly as $ab + cd$, which has ordinary length 5, whereas there is no corresponding way to simplify the expression $(a + b)(c + d)$, which is twice as long. The other two measures are more

robust in this respect. In particular, reverse polish length is a faithful measure for the amount of memory required to store the parse tree of a regular expression, and alphabetic width is often used in proofs of upper and lower bounds, compare [23]. A drawback of alphabetic width is that it may be far from the “real” size of a given regular expression. As an example, the expression $((\varepsilon + \emptyset) * \emptyset + \varepsilon) *$ has alphabetic width 0.

However, these three measures are all essentially identical, up to a constant multiplicative factor. We say “essentially” because one can always artificially inflate the ordinary length of a regular expression by adding arbitrarily many multiplicative factors of ε , additive factors of \emptyset , etc. In order to avoid such trivialities, we define what it means for a regular expression to be collapsible, as follows:

Definition 2.1. Let E be a regular expression over the alphabet Σ , and let $L(E)$ be the language specified by E . We say E is *collapsible* if any of the following conditions hold:

- (1) E contains the symbol \emptyset and $|E| > 1$;
- (2) E contains a subexpression of the form FG or GF where $L(F) = \{\varepsilon\}$;
- (3) E contains a subexpression of the form $F + G$ or $G + F$ where $L(F) = \{\varepsilon\}$ and $\varepsilon \in L(G)$.

Otherwise, if none of the conditions hold, E is said to be *uncollapsible*.

Definition 2.2. If E is an uncollapsible regular expression such that

- (1) E contains no superfluous parentheses; and
- (2) E contains no subexpression of the form F^{**} .

then we say E is *irreducible*.

Note that a minimal regular expression for E is uncollapsible and irreducible, but the converse does not necessarily hold. In [11] the following theorem is proved (cf. [25]).

Theorem 2.1. Let E be a regular expression over Σ . Let $|E|$ denote its ordinary length, let $|\text{rpn}(E)|$ denote its reverse polish length, and let $|\text{alph}(E)|$ denote the number of alphabetic symbols contained in E . Then we have

- (a) $|\text{alph}(E)| \leq |E|$;
- (b) If E is irreducible and $|\text{alph}(E)| \geq 1$, then $|E| \leq 11 \cdot |\text{alph}(E)| - 4$;
- (c) $|\text{rpn}(E)| \leq 2 \cdot |E| - 1$;
- (d) $|E| \leq 2 \cdot |\text{rpn}(E)| - 1$;
- (e) $|\text{alph}(E)| \leq \frac{1}{2}(|\text{rpn}(E)| + 1)$;
- (f) If E is irreducible and $|\text{alph}(E)| \geq 1$, then $|\text{rpn}(E)| \leq 7 \cdot |\text{alph}(E)| - 2$.

3 A simple grammar for valid regular expressions

As we have seen, if we want to enumerate regular expressions by size, we first have to agree upon a notion of expression size. But even then there still remains some ambiguity about the definition of a valid regular expression. For example, does the empty expression, that is, a string of length zero, constitute a valid regular expression? How about $()$ or

a^*a^* ? The first two, for example, generate errors in the software package Grail version 2.5 [44]. Surprisingly, very few textbooks, if any, define valid regular expressions properly or formally. For example, using the definition given in Martin [38, p. 86], the expression 00 is not valid, since it is not fully parenthesized. (To be fair, after the definition it is implied that parentheses can be omitted in some cases, but no formal definition of when this can be done is given.) Probably the best way to define valid regular expressions is with a grammar. We now present an unambiguous grammar for all valid regular expressions:

$$\begin{aligned}
S &\rightarrow E_+ \mid E_\bullet \mid G \\
E_+ &\rightarrow E_+ + F \mid F + F \\
F &\rightarrow E_\bullet \mid G \\
E_\bullet &\rightarrow E_\bullet G \mid GG \\
G &\rightarrow E_* \mid C \mid P \\
C &\rightarrow \emptyset \mid \varepsilon \mid a \quad (a \in \Sigma) \\
E_* &\rightarrow G^* \\
P &\rightarrow (S)
\end{aligned}$$

This grammar can be proved unambiguous by induction on the size of the regular expression generated. The meaning of the variables is as follows:

- S generates all regular expressions
- E_+ generates all unparenthesized expressions where the last operator was $+$
- E_\bullet generates all unparenthesized expressions where the last operator was \cdot (implicit concatenation)
- E_* generates all unparenthesized expressions where the last operator was $*$ (Kleene closure)
- C generates all unparenthesized expressions where there was no last operator (i.e., the constants)
- P generates all parenthesized expressions

Here by “parenthesized” we mean there is at least one pair of enclosing parentheses. Note this grammar allows a^*a^* , but disallows $()$. Once we have an unambiguous grammar, we can use a powerful tool — the Chomsky-Schützenberger theorem — to enumerate the number of expressions of size n .

4 Unambiguous context-free grammars and the Chomsky-Schützenberger theorem

Our principal tool for enumerating the number of strings of length n generated by an unambiguous context-free grammar is the Chomsky-Schützenberger theorem [4]. To state the theorem, we first recall some basic notions about grammars; these can be found in any introductory textbook on formal language theory, such as [24].

A *context-free grammar* is a quadruple of the form $G = (V, \Sigma, P, S)$, where V is a nonempty finite set of variables, Σ is a nonempty finite set called the *alphabet*, P is a finite subset of $V \times (V \cup \Sigma)^*$ called the *productions*, and $S \in V$ is a distinguished variable called the *start variable*. The elements of Σ are often called terminals. A production (A, γ) is typically written $A \rightarrow \gamma$. A *sentential form* is an element of $(V \cup \Sigma)^*$. Given a sentential form $\alpha A \beta$, where $A \in V$ and $\alpha, \beta \in (V \cup \Sigma)^*$, we can apply the production $A \rightarrow \gamma$ to get a new sentential form $\alpha \gamma \beta$. In this case we write $\alpha A \beta \Rightarrow \alpha \gamma \beta$. We write \Rightarrow^* for the reflexive, transitive closure of \Rightarrow ; that is, we write $\alpha \Rightarrow^* \beta$ if we can get from α to β by 0 or more applications of \Rightarrow . The language generated by a context-free grammar is the set of all strings of terminals obtained in 0 or more derivation steps from S , the start variable. Formally, $L(G) = \{x \in \Sigma^* : S \Rightarrow^* x\}$. A language is said to be *context-free* if it is generated by some context-free grammar. Given a sentential form α derivable from a variable A , we can form a *parse tree* for α as follows: the root is labeled A . Every node labeled with a variable B has subtrees with roots labeled, from left to right, with the elements of γ , where $B \rightarrow \gamma$ is a production. A grammar is said to be *unambiguous* if for each $x \in L(G)$ there is exactly one parse tree for x ; otherwise it is said to be *ambiguous*. It is known that not every context-free language has an unambiguous grammar.

Now we turn to formal power series; for more information, see, for example [51]. A formal power series over a commutative ring R in an indeterminate x is an infinite sequence of coefficients (a_0, a_1, a_2, \dots) chosen from R , and usually written $a_0 + a_1x + a_2x^2 + \dots$. The set of all such formal power series is denoted $R[[x]]$. The set of all formal power series is itself a commutative ring, with addition defined term-by-term, and multiplication defined by the usual Cauchy product as follows: if $f = a_0 + a_1x + a_2x^2 + \dots$ and $g = b_0 + b_1x + b_2x^2 + \dots$, then $fg = c_0 + c_1x + c_2x^2 + \dots$, where $c_n = \sum_{i+j=n} a_i b_j$. Exponentiation of formal series is defined, as usual, by iterated multiplication, so that $f^2 = ff$, for example. A formal power series f is said to be *algebraic* (over $R(x)$) if there exist a finite number of polynomials with coefficients in R , $r_0(x), r_1(x), \dots, r_n(x)$ such that

$$r_0(x) + r_1(x)f + \dots + r_n(x)f^n = 0.$$

The simplest nontrivial examples of algebraic formal series are the *rational functions*, which are quotients of polynomials $p(x)/q(x)$. Here is a less trivial example. The generating function of the Catalan numbers

$$f(x) = \sum_{n \geq 0} \frac{\binom{2n}{n}}{n+1} x^{n+1} = x + x^2 + 2x^3 + 5x^4 + 14x^5 + 42x^6 + 132x^7 + \dots,$$

is well known [49] to satisfy $f(x) = \frac{1}{2}(1 - \sqrt{1 - 4x})$, and hence we have $f^2 - f + x = 0$. Thus $f(x)$ is an algebraic (even quadratic!) formal series.

Now that we have the preliminaries, we can state the Chomsky-Schützenberger theorem:

Theorem 4.1. *If L is a context-free language having an unambiguous grammar, and $a_n := |L \cap \Sigma^n|$, then $\sum_{n \geq 0} a_n x^n$ is a formal power series in $\mathbb{Z}[[x]]$ that is algebraic over $\mathbb{Q}(x)$.*

Furthermore, the equation of which the formal power series is a root can be deduced as follows: first, we carry out the following replacements:

- Every terminal is replaced by a variable x
- Every occurrence of ε is replaced by the integer 1
- Every occurrence of \rightarrow is replaced by $=$
- Every occurrence of $|$ is replaced by $+$

By doing so, we get a system of algebraic equations, called the “commutative image” of the grammar, which can then be solved to find a defining equation for the power series. Oddly enough, Chomsky and Schützenberger did not actually provide a proof of their theorem. A proof is given by Kuich and Salomaa [32] and, more recently, by Panholzer [41].

Let’s look at a simple example. Consider the unambiguous grammar

$$\begin{aligned} S &\rightarrow M | U \\ M &\rightarrow 0M1M | \varepsilon \\ U &\rightarrow 0S | 0M1U \end{aligned}$$

which represents strings of “if-then-else” clauses. Then this grammar has the following commutative image:

$$S = M + U \tag{4.1}$$

$$M = x^2 M^2 + 1 \tag{4.2}$$

$$U = Sx + x^2 MU \tag{4.3}$$

This system of equations has the following power series solutions:

$$\begin{aligned} M &= 1 + x^2 + 2x^4 + 5x^6 + 14x^8 + 42x^{10} + \dots \\ U &= x + x^2 + 3x^3 + 4x^4 + 10x^5 + 15x^6 + 35x^7 + 56x^8 + \dots \\ S &= 1 + x + 2x^2 + 3x^3 + 6x^4 + 10x^5 + 20x^6 + 35x^7 + \dots \end{aligned}$$

By the Chomsky-Schützenberger theorem, each variable satisfies an algebraic equation over $\mathbb{Q}(x)$. We can solve the system above to find the equation for S , as follows: first, we solve (4.3) to get $U = \frac{Sx}{1-x^2M}$, and substitute back in (4.1) to get $S = M + \frac{Sx}{1-x^2M}$. Multiplying through by $1 - x^2M$ gives $S - x^2MS = M - x^2M^2 + Sx$, which, by (4.2), is equivalent to $S - x^2MS = 1 + Sx$. Solving for S , we get $S = \frac{1}{1-x^2M-x}$. Now (whatever M and x are) we have

$$(1 - x^2M - x)^2 = x^2(1 - M + x^2M^2) - x(2x - 1) - (2x - 1)(1 - x^2M - x),$$

so we get $S^{-2} = -x(2x - 1) - (2x - 1)S^{-1}$ and hence

$$x(2x - 1)S^2 + (2x - 1)S + 1 = 0.$$

This is an equation for S .

5 Solving algebraic equations using Gröbner bases

Before introducing the notion of Gröbner bases, we describe some of the relevant mathematical notions from the field of *commutative algebra*. The exposition here is impressionistic; readers familiar with algebraic geometry will have no difficulty reformulating it in more formalized terms. For readers seeking for a more thorough introduction into the topic, there are accessible textbooks at the undergraduate level, such as [5]; a standard graduate level textbook is [22].

We recall that a *field* k is a commutative ring with the additional property that multiplicative inverses exist. That is, for any non-zero $a \in k$, there exists an element b such that $ab = ba = 1$; more informally, one can “divide by a ”. Familiar examples of fields are the rational numbers \mathbb{Q} , the real numbers \mathbb{R} , and the complex numbers \mathbb{C} . On the other hand, the commutative ring \mathbb{Z} of integers is not a field, and the smallest field containing it is \mathbb{Q} .

For our application to the asymptotic enumeration of regular languages, we are interested in the commutative ring of formal power series $\mathbb{Z}[[x]]$. This is not a field, but rather only a ring — note, for example, that the element $2x$ does not have a multiplicative inverse. For the purposes of our algebraic framework it is convenient to work with the field $k = \mathbb{Q}((x))$ of formal Laurent series over \mathbb{Q} . A formal Laurent series is defined similarly to a formal power series, with the difference that finitely many negative exponents are allowed; an example is

$$\frac{e^x}{x^2} = \frac{1}{x^2} + \frac{1}{x} + \frac{1}{2} + \frac{x}{6} + \frac{x^2}{24} + \cdots.$$

The following discussion holds for any field k , but for intuition, the reader may prefer to think of $k = \mathbb{R}$.

Given any field k and indeterminates X_1, X_2, \dots, X_n , there are two important objects:

- the n -dimensional vector space $W = k^n$ over k , with coordinates X_i ($1 \leq i \leq n$); and
- the ring $k[X_1, X_2, \dots, X_n]$ of (multivariate) polynomials over k in n indeterminates.

For instance, taking $k = \mathbb{Q}((x))$, the polynomial $Sx + x^2MU - U$, which we used in the previous section in Equation (4.3), is member of the ring $k[S, M, U]$. The corresponding vector space W has coordinates S, M , and U . Notice that x is not a coordinate of W , but an artifact originating from the way the members of k are defined.

Given any collection of polynomials \mathcal{F} in R , we can define their *vanishing set* $V(\mathcal{F})$ to be the set of common solutions in W ; that is, all points $(x_1, x_2, \dots, x_n) \in W$ such that

$$f(x_1, x_2, \dots, x_n) = 0 \quad \text{for all } f \in \mathcal{F}.$$

As an example, let $W = \mathbb{R}^3$, with coordinates X, Y, Z . Then, the vanishing set of the set of polynomials $\mathcal{F} = \{X, Y + 3, Z + Y - 2\}$ is the single point given by $(X, Y, Z) = (0, -3, 5)$; the vanishing set of the single polynomial $Z - X^2 - Y^2$ is an upward-opening paraboloid.

The *ideal* $\langle \mathcal{F} \rangle$ generated by a collection \mathcal{F} of polynomials is the set of all R -linear

combinations of \mathcal{F} ; that is, all polynomials of the form

$$p_1 \cdot f_1 + p_2 \cdot f_2 + \cdots + p_\ell \cdot f_\ell \quad \text{where } p_i \in R, f_i \in \mathcal{F} \text{ for all } i.$$

Observe that the vanishing sets of a collection of polynomials and their generated ideal are equal: $V(\mathcal{F}) = V(\langle \mathcal{F} \rangle)$.

A *term ordering* on R is a total order \prec on the set of monomials (disregarding coefficients) of R satisfying

- *multiplicativity* — if u, v, w are any monomials in R , then $u \prec v$ implies $wu \prec wv$;
- *well-ordering* — if \mathcal{F} is a collection of monomials, then \mathcal{F} has a smallest element under \prec .

Once a term ordering has been defined, one can then define the notion of the *leading term* of a polynomial, similar to the univariate case. For example, one defines the *pure lexicographic order* on $k[X, Y, Z]$ given by $Z \prec Y \prec X$ to be the ordering where $X^a Y^b Z^c \prec X^d Y^e Z^f$ if and only if $(a, b, c) < (d, e, f)$ lexicographically. With this ordering, an example of a polynomial with its monomials in decreasing order is

$$X^3 + X^2 Y + X^2 Z^7 + Y^9 + 1;$$

its *leading term* is $X^3 = X^3 Y^0 Z^0$, and its *trailing terms* are $X^2 Y$, $X^2 Z^7$, Y^9 and 1.

Given an ideal I , a *Gröbner basis* \mathcal{B} for I is a set of polynomials g_1, g_2, \dots, g_k such that the ideal generated by the leading terms of the g_i is precisely the initial ideal of I , defined to be the set of leading terms of polynomials in I . It can be shown that \mathcal{B} generates I . Furthermore, we say that \mathcal{B} is a *reduced Gröbner basis* if

- the coefficient of each leading term in \mathcal{B} is 1;
- the leading terms of \mathcal{B} are a *minimal* set of generators for the initial ideal of I ; and
- no trailing terms of \mathcal{B} appear in the initial ideal of I .

Once a term order has been chosen, reduced Gröbner bases are unique. Note that in general, there are many term orderings for a polynomial ring R ; the computational difficulty of a computation involving Gröbner bases is often highly sensitive to the choice of term ordering used.

Having established these preliminaries, we turn our attention to solving a system of equations given by the commutative image of a context free grammar. Suppose we have a context-free grammar in the non-terminals S, N_1, N_2, \dots, N_n . For each non-terminal N , let f_N also denote the generating function enumerating the language generated by N . Taking k to be the field of formal Laurent series $\mathbb{Q}((x))$, the Chomsky-Schützenberger theorem implies $f_N \in k$ for every non-terminal N . Furthermore, by taking the commutative image of the context-free grammar, we obtain a sequence of polynomials $p_S, p_{N_1}, \dots, p_{N_n}$, where for every non-terminal N , the polynomial relation p_N is the commutative image of the derivation rule for N . Note that every such polynomial is in the polynomial ring $(\mathbb{Z}[x])[S, N_1, N_2, \dots, N_n]$.

It follows from the definitions that for every non-terminal N ,

$$p_N(f_S, f_{N_1}, f_{N_2}, \dots, f_{N_n}) = 0;$$

that is, the $(n+1)$ -tuple $(f_S, f_{N_1}, f_{N_2}, \dots, f_{N_n})$ is a zero of the polynomial p_N . Since this holds for every non-terminal N , we can equivalently say that $(f_S, f_{N_1}, f_{N_2}, \dots, f_{N_n})$ is in the vanishing set $V(I)$, where I is generated by the polynomials $p_S, p_{N_1}, p_{N_2}, \dots, p_{N_n}$.

Our aim is to determine an algebraic equation satisfied by the power series f_S . To do this, we find a Gröbner basis \mathcal{B} for I , using an *elimination ordering* on the indeterminate S . The defining property of any such term ordering is that the monomials involving only the indeterminate S are strictly smaller than the other monomials; namely, those involving at least one of N_1, N_2, \dots, N_n . By the Chomsky-Schützenberger theorem and the properties of Gröbner bases, the smallest polynomial p in \mathcal{B} will be a univariate polynomial in the indeterminate S . Since $p \in I$, and $(f_S, f_{N_1}, f_{N_2}, \dots, f_{N_n})$ is in the vanishing set $V(I)$, we see that $p(f_S) = 0$; that is, $p = 0$ is an algebraic equation satisfied by f_S . (Note that in previous sections, we simply use S to denote f_S .)

As an example, we use Maple 13 to compute such an algebraic equation for the example grammar in the previous section. We give the commands, followed by the produced output. The commutative image of the grammar is entered as a list of polynomials, given by

```
> eqs := [ -S + M + U, -M + x^2*M^2 + 1, -U + S*x + x^2*M*U ];
```

$$eqs := [-S + M + U, -M + x^2 M^2 + 1, -U + Sx + x^2 MU].$$

Maple provides an elimination ordering called `lexdeg`; to compute a reduced Gröbner basis using this ordering, we enter the command

```
> Groebner[Basis](eqs, lexdeg([M, U], [S]));
```

$$[1 + (-1 + 2x)S + (-x + 2x^2)S^2, 1 + (-1 + x)S + Ux, -1 + (1 - 2x)S + Mx].$$

The algebraic equation satisfied by S is the first polynomial in this set:

```
> algeq := %[1];
```

$$algeq := 1 + (-1 + 2x)S + (-x + 2x^2)S^2.$$

To compute the Laurent series zeros of S using this polynomial, we solve for S and expand the solutions as Laurent series in the indeterminate x :

```
> map(series, [solve(algeq, S)], x);
```

$$[(-x^{-1} - 1 - x - 2x^2 - 3x^3 - 6x^4 - 10x^5 + O(x^6)), (1 + x + 2x^2 + 3x^3 + 6x^4 + 10x^5 + O(x^6))].$$

Our desired power series solution is the second entry in the above returned list.

6 Asymptotic bounds via singularity analysis

If L is a context-free language having an unambiguous grammar and $f(x) = \sum a_n x^n$ is the formal power series enumerating it, then $f(x)$ is algebraic over $\mathbb{Q}(x)$ by Theorem 4.1. The previous section gave a procedure for computing an algebraic equation satisfied by f ; that is, we are able to determine a non-trivial polynomial $P(x, S) \in \mathbb{Z}[x, S]$ such that $P(x, f(x)) = 0$. This section describes how *singularity analysis* can be used to determine the asymptotic growth rate of the coefficients a_n . We sketch some of the requisite notions from complex analysis and provide a glimpse of the underlying theory; more details can be found in Flajolet and Sedgewick [12].

The usefulness in considering complex analysis is that the formal power series $f(x)$, defined purely combinatorially, can be viewed as a function defined on an appropriate open subset of the complex plane \mathbb{C} . Such a function is called *holomorphic* or (*complex*) *analytic*; this reinterpretation of $f(x)$ allows us to apply theorems from complex analysis in order to derive bounds on the asymptotic growth rate of the a_n far tighter than what we could do with purely combinatorial reasoning.

Indeed, assume that L is an infinite context-free language — then there exists a real number $0 < R \leq 1$ called the *radius of convergence* for $f(x)$. The defining properties of R are that:

- if z is a complex number with $|z| < R$, then the infinite sum $a_0 + a_1z + a_2z^2 + a_3z^3 + \dots$ converges; and
- if z is a complex number with $|z| > R$, then the infinite sum $a_0 + a_1z + a_2z^2 + a_3z^3 + \dots$ diverges.

We note that the definition says nothing about the convergence of $\sum a_i z^i$ when $|z| = R$. Thus, defining U to be the open ball of complex numbers z satisfying $|z| < R$, we can reinterpret f as an *analytic function on U* . The connection between the asymptotic growth of the coefficients a_n and the number R is given by two theorems.

Theorem 6.1 (Hadamard). *Given any power series, R is given by the explicit formula:*

$$R = \frac{1}{\limsup_{n \rightarrow \infty} |a_n|^{1/n}}.$$

The defining properties of \limsup state that

- for any $\varepsilon > 0$, the relation $|a_n|^{1/n} < \frac{1}{R} + \varepsilon$ holds for sufficiently large n ; and
- for any $\varepsilon > 0$, the relation $|a_n|^{1/n} > \frac{1}{R} - \varepsilon$ holds for infinitely many n .

For our situation in particular, this implies that up to a sub-exponential factor, a_n grows asymptotically like $1/R^n$. (This implies that for any $\varepsilon > 0$, we have $a_n \in O((\frac{1}{R} + \varepsilon)^n)$ and $a_n \notin O((\frac{1}{R} - \varepsilon)^n)$).

We note that Hadamard's formula applies to *any* power series, not just to generating functions of context-free languages.

An elementary argument shows that our assumption that L is infinite implies $R \leq 1$; similarly, our assumption that L is context-free (and thus algebraic) implies $R > 0$. (The argument for showing $R > 0$ is harder, and is sketched here for those familiar with complex analysis. The algebraic curve given by $P(z, y) = 0$ determines d branches around $z = 0$ and the power series $f(x) = \sum_n a_n x^n$ must be associated with one such branch. Since the exponents of $f(x)$ are non-negative integers, this must be an analytic branch at 0; hence, $f(x)$ determines an analytic function at 0 and must have positive radius of convergence.)

The second theorem describes the convergence of the power series $f(x)$ on the circle given by $|z| = R$. A *dominant singularity* for $f(x)$ is a point z_0 on this circle such that the sum $\sum a_n z_0^n$ diverges; the following result says that a positive (real-valued) dominant singularity always exists.

Theorem 6.2 (Pringsheim). *Let $f(x) = \sum_n a_n x^n$ be a power series with radius of convergence $R > 0$. If the coefficients a_n are all non-negative, then R is a dominant singularity for $f(x)$.*

The benefit of Pringsheim's theorem is that, for the sake of determining R , it suffices to examine the positive real line for the singularities of $f(x)$ considered as a *function*, not just as a power series. We make this more precise now, by introducing the concept of a *multi-valued function*.

Suppose that the power series $f(x)$ is algebraic of degree d over $\mathbb{Q}(x)$ — under the assumption that P is irreducible, this means that the degree of the polynomial $P(x, S) \in \mathbb{Z}[x, S]$ in the variable S is d , and we may write

$$P = q_n S^n + q_{n-1} S^{n-1} + q_{n-2} S^{n-2} + \cdots + q_0,$$

where each q_i is a polynomial in $\mathbb{Z}[x]$ and q_n is non-zero. (If P is reducible, factor it and replace it by an appropriate irreducible factor.)

If we work in the algebraically closed *Puiseux series field* $\bigcup_{n \geq 1} \mathbb{C}((x^{1/n}))$, we obtain d roots of $P(x, S) = 0$, say, $g_1(x), g_2(x), \dots, g_d(x)$, one of which coincides with $f(x)$. In general, these roots will not be power series with non-negative integer coefficients, but instead will be more generalized power series with complex coefficients and (possibly negative) fractional exponents.

Let $D(x) \in \mathbb{Z}[x]$ be the *discriminant* of P with respect to the variable S ; this is readily computed via the formula

$$D = \frac{(-1)^{n(n-1)/2}}{q_n} \cdot \text{Res}\left(P, \frac{\partial}{\partial S} P, S\right).$$

Here, Res denotes the *resultant* of two polynomials, defined to be the determinant of a matrix whose entries are given by the coefficients of the polynomials. The theoretical importance of D is that it satisfies the identity

$$D(x) = q_n^{2(n-1)} \prod_{i \neq j} (g_i(x) - g_j(x)).$$

Define the *exceptional set* Ξ of P to be the complex zeros of D ; note that this is a finite set. For every point z in the complement $\mathbb{C} \setminus \Xi$, where D does not vanish, there exist d distinct solutions y to the equation $P(z, y) = 0$. Furthermore, the d distinct solutions vary continuously with z , and a locally continuous choice of solutions locally determines a *branch* (which is locally an analytic function) of the algebraic curve cut out by $P(z, y) = 0$; this is how a *multi-valued function* arises.

On the open set U , which we have defined to be the set of points z satisfying $|z| < R$, one such branch is given by our initial power series $f(x)$. By Pringsheim's theorem, $f(x)$ diverges at R ; this shows that $f(x)$, considered as an analytic function on U , has no analytic continuation to a function on an open set containing $U \cup \{R\}$. According to the discussion above, this shows that R must be in the exceptional set Ξ .

We have given a method to calculate an upper bound for the growth rate of the a_n ; in particular, we have shown parts (1) and (2) of:

Theorem 6.3. *Let $f(x) = \sum_n a_n x^n$ be a formal power series where $a_n \geq 0$ for each n . Suppose $P(x, S) = 0$ is a non-trivial algebraic equation satisfied by $f(x)$, and let D be the discriminant of P with respect to S . Then, exactly one of the positive real roots R of D satisfies the following properties:*

- (1) for any $\varepsilon > 0$, $a_n \in O((\frac{1}{R} + \varepsilon)^n)$;

- (2) for any $\varepsilon > 0$, $a_n \notin O((\frac{1}{R} - \varepsilon)^n)$; and
 (3) if D has no zero $z_0 \neq R$ such that $|z_0| = R$, then for any $\varepsilon > 0$, $a_n \in \Omega((\frac{1}{R} - \varepsilon)^n)$.

We remark that part (3) is much more difficult to show; it is implied by the stronger result that if D has no zero $z_0 \neq R$ such that $|z_0| = R$, then there exists a polynomial p such that $a_n \sim p(n) \cdot (\frac{1}{R})^n$.

Given the list $\rho_1 < \rho_2 < \dots < \rho_k$ of positive real-valued elements of Ξ , there remains the task of selecting which ρ_j to use to provide an upper or lower bound. The bigger j is, the better our upper bound will be; however, for this bound to be valid, we must ensure that $\rho_j \leq R$. For our purposes, we simply employ a boot-strapping method — if it is known beforehand that $a_n \in O(n^s)$ for some s , then we simply choose the minimal j such that $1/\rho_j \leq s$; equivalently, $\rho_j \geq 1/s$. If this is not possible, we simply pick $j = 1$. Similarly, for a lower bound, we choose the maximal j such that $\rho_j \leq 1/t$ if it is known that $a_n \in \Omega(n^t)$. (With much more work, one can precisely identify R — Flajolet and Sedgewick [12] describe an algorithm “Algebraic Coefficient Asymptotics” that does this.)

As an illustration, we continue the Maple example in the previous section to derive an asymptotic upper bound for the example grammar. We first recall the algebraic equation satisfied by S :

```
> algeq;
```

$$1 + (-1 + 2x)S + (-x + 2x^2)S^2.$$

We compute the discriminant D :

```
> d := discrim(algeq, S);
```

$$d := -(2x + 1)(-1 + 2x).$$

The real roots of D are given by:

```
> realroots := [fsolve(%)];
```

$$\text{realroots} := [-0.5000000000, 0.5000000000].$$

Finally, an upper bound is given by taking the inverse of the smallest positive real root:

```
> 1/min(op(select(type, realroots, positive)));
```

$$2.000000000.$$

Hence, $a_n \in O((2 + \varepsilon)^n)$ for any $\varepsilon > 0$.

7 Lower bounds on enumeration of regular languages by regular expressions

We now turn to lower bounds on $R_k(n)$. In the unary case ($k = 1$), we can argue as follows: consider any subset of $\{\varepsilon, a, a^2, \dots, a^{t-1}\}$. Such a subset can be denoted by a regular expression of (ordinary) length at most $t(t + 1)/2$. Since there are 2^t distinct

subsets, this gives a lower bound of $R_1(n) \geq 2^{\sqrt{2n}-1}$. Similarly, when $k \geq 2$, there are k^n distinct strings of length n , so $R_k(n) \geq k^n$. These naive bounds can be improved somewhat using a grammar-based approach.

Consider a regular expression of the form

$$w_1(\varepsilon + w_2(\varepsilon + w_3(\varepsilon + \dots)))$$

where the w_i denote nonempty words. Every distinct choice of the w_i specifies a distinct language. Such expressions can be generated by the grammar

$$\begin{aligned} S &\rightarrow Y \mid Y(\varepsilon + S) \\ Y &\rightarrow aY \mid a, \quad a \in \Sigma \end{aligned}$$

which has the commutative image

$$\begin{aligned} S &= Y + YSx^4 \\ Y &= kxY + kx. \end{aligned}$$

The solution to this system is

$$S = \frac{kx}{1 - kx - kx^5}.$$

Once again, the asymptotic behavior of the coefficients of the power series for S depend on the zeros of $1 - kx - kx^5$. The smallest (indeed, the only) real root is, asymptotically as $k \rightarrow \infty$, given by

$$\sum_{i \geq 0} \frac{(-1)^i \binom{5i}{i}}{4i + 1} k^{-(4i+1)} = \frac{1}{k} - \frac{1}{k^5} + \frac{5}{k^9} - \frac{35}{k^{13}} + \dots$$

The reciprocal of this series is

$$\sum_{i \geq 0} \frac{4 \binom{5i+5}{i+1}}{5(5i+4)} k^{1-4i} = k + \frac{1}{k^3} - \frac{4}{k^7} + \frac{26}{k^{11}} - \frac{204}{k^{15}} + \frac{1771}{k^{19}} - \dots$$

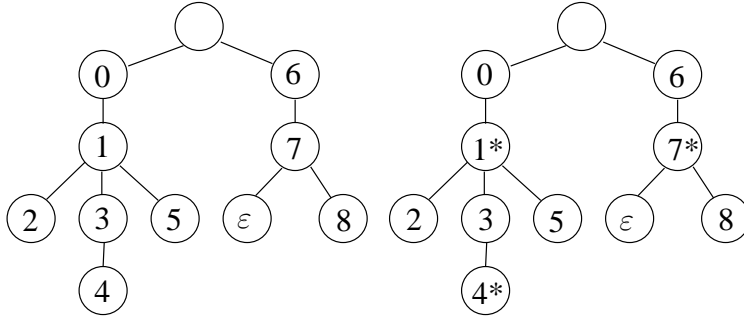
For $k = 1$ the only real root of $1 - kx - kx^5$ is approximately .754877666 and for $k = 2$ it is about .4756527435. Thus we have

Theorem 7.1. $R_1(n) = \Omega(1.3247^n)$ and $R_2(n) = \Omega(2.102374^n)$.

7.1 Trie representations for finite languages

We will now improve these lower bounds. To this end, we begin with the simpler problem of counting the number of finite languages that may be specified by regular expressions without Kleene star of size n . Non-empty finite languages not containing ε admit a standard representation via a trie structure; an example is given Fig. 1(a).

The words in such a language L correspond to the leaf nodes of the trie for L ; moreover, the concatenation of labels from the root to a leaf node gives an expression for the word associated with that leaf node. For regular languages L and M , we write $M^{-1}L$ to



(a) Representing the finite language $01(2+34+5)+67(\varepsilon+8)$ as a trie.
 (b) Representing the infinite language $01*(2+34*+5)+67*(\varepsilon+8)$ as a starred trie.

Figure 1. Example of a trie representation for a finite language (see Section 7.1) and of a starred trie representation for an infinite language (see Section 7.2).

denote the left quotient of L by M ; formally

$$M^{-1}L = \{v : \text{there exists } u \in M \text{ such that } uv \in L\}.$$

If M consists of a single word w , we also write $w^{-1}L$ instead of $\{w\}^{-1}L$, and $w^{-n}L$ instead of $(w^n)^{-1}L$.

For notational convenience, we take our alphabet to be $\Sigma = \{a_0, a_1, \dots, a_{k-1}\}$, where $k \geq 1$ denotes our alphabet size. A trie encodes the simple fact that each nonempty finite language L not containing ε can be uniquely decomposed as $L = \bigcup_i a_i L_i$, where $L_i = a_i^{-1}L$, and the index i runs over all symbols $a_i \in \Sigma$ such that L_i is nonempty. This factoring out of common prefixes resembles Horner's rule (see e.g. [26, p. 486]) for evaluating polynomials. We develop lower bounds by specifying a context-free grammar that generates regular expressions with common prefixes factored out. In fact, the grammar is designed so that if r is a regular expression generated by the grammar, then the structure of r mimics that of the trie for $L(r)$ — nodes with a single child correspond to concatenations, while nodes with multiple children correspond to concatenations with a union, see Table 1.

$S \rightarrow$	$Y \mid Z$
$E \rightarrow$	$Y \mid (Z) \mid (\varepsilon + S)$
$Y \rightarrow$	$P_i \text{ for } 0 \leq i < k$
$Z \rightarrow$	$P_{n_0} + P_{n_1} + \dots + P_{n_t}$
	where $0 \leq n_0 < n_1 < \dots < n_t < k$ for $t > 0$
$P_i \rightarrow$	$a_i \mid a_i E \text{ for } 0 \leq i < k$

Table 1. A grammar for mimicking tries with regular expressions.

The set of regular languages represented corresponds to all non-empty finite languages over Σ not containing the empty string ε . We briefly describe the non-terminals:

	ordinary	reverse polish	alphabetic
1	$\Omega(1.3247^n)$	$\Omega(1.2720^n)$	$\Omega(2^n)$
2	$\Omega(2.5676^n)$	$\Omega(2.1532^n)$	$\Omega(6.8284^n)$
3	$\Omega(3.6130^n)$	$\Omega(2.7176^n)$	$\Omega(11.1961^n)$
4	$\Omega(4.6260^n)$	$\Omega(3.1806^n)$	$\Omega(15.5307^n)$
5	$\Omega(5.6264^n)$	$\Omega(3.5834^n)$	$\Omega(19.8548^n)$
6	$\Omega(6.6215^n)$	$\Omega(3.9451^n)$	$\Omega(24.1740^n)$

Table 2. Lower bounds for $R_k(n)$ with respect to size measure and alphabet size.

S generates all non-empty finite languages not containing ε .

E generates all non-empty finite languages containing at least one word other than ε .

Y generates all non-empty finite languages (not containing ε) whose words all begin with the same letter. The `for` loop is executed only once.

Z generates all non-empty finite languages (not containing ε) whose words do not all begin with the same letter.

P_i generates all non-empty finite languages (not containing ε) whose words all begin with a_i .

We remark that this grammar is unambiguous and that no regular language is represented more than once; this should be clear from the relationship between regular expressions generated by the grammar and their respective tries.

(Note that it is possible to slightly optimize this grammar in the case of ordinary length to generate expressions such as $0 + 00$ in lieu of $0(\varepsilon + 0)$, but as it results in marginal improvements to the lower bound at the cost of greatly complicating the grammar, we do not do so here.)

Table 2 lists the lower bounds obtained through this grammar. In this table (and only this table), each $\Omega(k^n)$ in the column corresponding to reverse polish notation should be interpreted as “not $O(k^n)$ ” — observe, for instance, that all strings produced by our grammar for a unary alphabet have odd reverse polish length.

Remark 7.2. Using the singularity analysis method explained in Section 6, these lower bounds were obtained by bootstrapping off the trivial bounds of $\Omega(k^n)$, $\Omega(k^{n/2})$ and $\Omega(k^n)$ for the ordinary, reverse polish length and alphabetic width cases, respectively.

Before we generalize our approach to cover also infinite languages, we derive a formula showing how our lower bound on alphabetic width will increase along with the alphabet size k .

To this end, we first state a version of the Lagrange implicit function theorem as a simplification of [14, Theorem 1.2.4]. If $f(x)$ is a power series in x , we write $[x^n]f(x)$ to denote the coefficient of x^n in $f(x)$; recall that the *characteristic* of a ring R with additive identity 0 and multiplicative identity 1 is defined to be the smallest integer k such that $\sum_{i=1}^k 1 = 0$, or zero if there is no such k .

Lemma 7.3. *Let R be a commutative ring of characteristic zero and take $\phi(\lambda) \in R[[\lambda]]$ such that $[\lambda^0]\phi$ is invertible. Then there exists a unique formal power series $w(x) \in R[[x]]$*

such that $[x^0]w = 0$ and $w = x\phi(w)$. For $n \geq 1$,

$$[x^n]w(x) = \frac{1}{n}[\lambda^{n-1}]\phi^n(\lambda).$$

Due to the simplicity of alphabetic width, the problem of enumerating regular languages in this case may be interpreted as doing so for rooted k -ary trees, where each internal node is marked with one of two possible colours. We thus investigate how our lower bound varies with k .

More specifically, consider a regular expression r generated by the grammar from the previous section and its associated trie. Colour each node with a child labelled ε black and all other nodes white. After deleting all nodes marked ε , call the resultant tree $T(r)$. This operation is reversible, and shows that we may put the expressions of alphabetic width n in correspondence with the k -ary rooted trees with $n + 1$ vertices where every non-root internal node may assume one of two colours. In order to estimate the latter, we first prove a basic result. The first half of the following lemma is also found in [12, p. 68].

Lemma 7.4. *There are $\frac{1}{n} \binom{kn}{n-1}$ k -ary trees of n nodes. Moreover, the expected number of leaf nodes among k -ary trees of n nodes is asymptotic to $(1 - 1/k)^k n$ as $n \rightarrow \infty$.*

Proof. Fix $k \geq 1$. For $n \geq 1$, let a_n denote the number of k -ary rooted trees with n vertices and consider the generating series:

$$f(x) = \sum_{n \geq 1} a_n x^n.$$

By the recursive structure of k -ary trees, we have the recurrence:

$$f(x) = t(1 + f(x))^k.$$

Thus, by the Lagrange implicit function theorem, we have

$$a_n = [x^n]f(x) = \frac{1}{n}[\lambda^{n-1}](1 + \lambda)^{kn} = \frac{1}{n} \binom{kn}{n-1}.$$

We now calculate the number of leaf nodes among all k -ary rooted trees with n vertices. Let $b_{n,m}$ denote the number of k -ary rooted trees with n vertices and m leaf nodes and c_n the number of leaf nodes among all k -ary rooted trees with n vertices. Consider the bivariate generating series:

$$g(x, y) = \sum_{n, m \geq 1} b_{n,m} x^n y^m.$$

By the recursive structure of k -ary trees, we have the recurrence:

$$g(x, y) = y(x - 1 + (1 + g(x, y))^k).$$

The Lagrange implicit function theorem once again yields

$$\begin{aligned}
c_n &= \left. \frac{\partial}{\partial x} [y^n] g(x, y) \right|_{x=1} \\
&= \left. \frac{\partial}{\partial x} \frac{1}{n} [\lambda^{n-1}] (x-1 + (1+g(x, y))^k)^n \right|_{x=1} \\
&= \frac{1}{n} [\lambda^{n-1}] \left. \frac{\partial}{\partial x} (x-1 + (1+\lambda)^k)^n \right|_{x=1} \\
&= [\lambda^{n-1}] (1+\lambda)^{k(n-1)} \\
&= \binom{k(n-1)}{n-1}.
\end{aligned}$$

Thus, the expected number of leaf nodes among n -node trees is, as $n \rightarrow \infty$ while having k fixed,

$$\frac{c_n}{a_n} = \frac{n \binom{k(n-1)}{n-1}}{\binom{kn}{n-1}} \sim n \left(\frac{k-1}{k} \right)^k.$$

□

We wish to find a bound on the expected number of subsets of non-root internal nodes among all k -ary rooted trees with n nodes, where a subset corresponds to those nodes marked black. Fix $k \geq 2$. Since the map $x \mapsto 2^x$ is convex, for every $\varepsilon > 0$ and sufficiently large n , Jensen's inequality (e.g., [47, Thm. 3.3]) applied to the lemma above implies the following lower bound on the number of subsets:

$$2^{(1-(1-1/k)^k - \varepsilon)n}.$$

Since $-(1-1/k)^k > -1/e$ for $k \geq 1$, we may choose $\varepsilon > 0$ such that

$$-(1-1/k)^k - \varepsilon > -1/e.$$

This yields a lower bound of $2^{(1-1/e)n}$.

Assuming $k \geq 2$ fixed, we now estimate $\binom{kn}{n-1}$. By Stirling's formula, we have, as $n \rightarrow \infty$,

$$\binom{kn}{n-1} = \Theta \left(\left(\frac{k^k}{(k-1)^{k-1}} \right)^n \right).$$

Putting our two bounds together, we have the following lower bound on the number of star-free regular expressions of alphabetic width n , when $n \rightarrow \infty$ while keeping k fixed:

$$\Omega \left(\left(\frac{2^{(1-1/e)} k^k}{(k-1)^{k-1}} \right)^n \right).$$

7.2 Trie representations for some infinite regular languages

We now turn our attention to enumerating regular languages in general; that is, we allow for regular expressions with Kleene stars.

Our grammars for this section are based on the those for the star-free cases. Due to the difficulty of avoiding specifying duplicate regular languages, we settle for a “small” subset of regular languages. For simplicity, we only consider taking the Kleene star closure of singleton alphabet symbols, and we impose some further restrictions.

Recall the trie representation of a star-free regular expression written in our common prefix notation. With this representation, we may mark nodes with stars while satisfying the following conditions:

- each starred symbol must have a non-starred parent other than the root;
- a starred symbol may not have a sibling or an identically-labelled parent (disregarding the lack of star) with its own sibling; and
- a starred symbol may not have an identically-labelled child (disregarding the lack of star).

The first condition eliminates duplicates such as

$$0*11*0*1*0* \leftrightarrow 0*1*0*11*0*;$$

the second eliminates those such as

$$01* \leftrightarrow 0(\varepsilon+11*) \text{ and } 0(1+2*1) \leftrightarrow 02*1$$

and the third eliminates those such as

$$0*0 \leftrightarrow 00*.$$

In this manner, we end up with starred tries such as in Fig. 1(b). Algorithm 1 illustrates how to recreate such a starred trie from the language it specifies.

Algorithm 1 STAR-TRIE(L)

Require: $\varepsilon \notin L, L \neq \emptyset$

- 1: create a tree T with unlabelled root
 - 2: **for all** $a \in \Sigma$ such that $a^{-1}L \neq \emptyset$ **do**
 - 3: append STAR-TRIE-HELP($a^{-1}L, a$) below the root of T
 - 4: **end for**
 - 5: return T
-

Let T be any starred trie satisfying the conditions above. Then T represents a regular expression, which in turn specifies a certain language. We now show that when the algorithm is run with that language as input, it returns the trie T by arguing that at each step of the algorithm when a particular node (matched with language L if the root and aL otherwise) is being processed, the children are correctly reconstructed.

We first consider children of the root. By the original trie construction (for finite languages without ε), no such children may be labelled ε . Thus, by the first star condition, the only children may be unstarred alphabet symbols. Thus, line 2 of Algorithm 1 suffices to find all children of the root correctly.

Algorithm 2 STAR-TRIE-HELP(L, a)

```

1: create a tree  $T$  with root labelled  $a$ 
2: for all  $b \in \Sigma$  s.t.  $b^{-1}L \neq \emptyset$  do
3:   if  $(b^{-n}L) \cap (\varepsilon + (\Sigma \setminus \{b\})\Sigma^*) \neq \emptyset$  for all  $n \geq 0$  then {need a child labelled  $b^*$ }
4:     append a new  $b^*$ -node below the root of  $T$ 
5:     if  $L \neq b^*$  then { $b^*$  will be an internal node}
6:       for all  $c \in \Sigma \setminus \{b\}$  such that  $c^{-1}L \neq \emptyset$  do {determine children of  $b^*$ }
7:         append STAR-TRIE-HELP( $c^{-1}L, c$ ) below the  $b^*$ -node
8:       end for
9:       if  $b \in L$  then
10:        append a new  $\varepsilon$ -node below the  $b^*$ -node
11:       end if
12:     end if
13:   else {need a child labelled  $b$ }
14:     append STAR-TRIE-HELP( $b^{-1}L, b$ ) below the root of  $T$ 
15:   end if
16: end for
17: if  $\varepsilon \in L$  and the root of  $T$  has at least one unstarred child then
18:   append a new  $\varepsilon$ -node below the root of  $T$ 
19: end if
20: return  $T$ 

```

Now consider a non-root internal node, say labelled a . By the third star condition, a starred node may not have a child labelled with the same alphabet symbol, so if a has a child labelled b^* , then

$$(b^n)^{-1}L \cap (\varepsilon + (\Sigma \setminus \{b\})\Sigma^*) \text{ is non-empty for all } n \geq 0. \quad (7.1)$$

Conversely, by the second condition, a starred node may not have an identically-labelled parent that has ε as a sibling, so if (7.1) holds, then a must have a child labelled b^* . By the second star condition, a starred node may not have siblings, so the algorithm need not check for other children once a starred child is found. This shows that line 3 of Algorithm 2 correctly identifies all starred children of a . Assuming a has a starred child b^* , then by the third condition, line 6 of Algorithm 2 correctly recovers all children of b^* . All remaining children of a have no stars, and line 14 of Algorithm 2 suffices to find all children labelled with $a \in \Sigma$; the special case of an ε -child below a is covered by line 17.

We give a grammar that generates expressions meeting these conditions in Table 3. As before, we take our alphabet to be $\Sigma = \{a_0, a_1, \dots, a_{k-1}\}$. We describe the roles of the non-terminals of the grammar in Table 3.

S generates all expressions — this corresponds to Algorithm 1.

E, E_i generate expressions that may be concatenated to non-starred and starred alphabet symbols, respectively. The non-terminal E corresponds to lines 2 and 13 while E_i corresponds to line 5 of Algorithm 2. These act the same as S except for the introduction of parentheses to take precedence into account and restriction that no prefixes of the form $\varepsilon + aa^*$ are generated, used to implement the second condition.

$S \rightarrow$	$Y \mid Z$
$E \rightarrow$	$Y \mid (Z) \mid (\varepsilon + Y') \mid (\varepsilon + Z)$
$E_i \rightarrow$	$Y_i \mid (Z_i) \mid (\varepsilon + Y'_i) \mid (\varepsilon + Z_i) \text{ for } 0 \leq i < k$
$Y \rightarrow$	$P_i \text{ for } 0 \leq i < k$
$Y' \rightarrow$	$P'_i \text{ for } 0 \leq i < k$
$Y_i \rightarrow$	$P_j \text{ for } 0 \leq i, j < k \text{ and } i \neq j$
$Y'_i \rightarrow$	$P'_j \text{ for } 0 \leq i, j < k \text{ and } i \neq j$
$Z \rightarrow$	$P'_{n_0} + P'_{n_1} + \dots + P'_{n_t}$ where $0 \leq n_0 < n_1 < \dots < n_t < k$ for $t > 0$
$Z_i \rightarrow$	$P'_{n_0} + P'_{n_1} + \dots + P'_{n_t}$ as above, but with $n_j \neq i$ for all $0 \leq j \leq t$
$P_i \rightarrow$	$a_i \mid a_i E \mid a_i a_j^* \mid a_i a_j^* E_j$ for $0 \leq i, j < k$
$P'_i \rightarrow$	$a_i \mid a_i E \mid a_i a_j^* \mid a_i a_j^* E_j$ for $0 \leq i, j < k$ and $i \neq j$

Table 3. A grammar generating all regular expressions meeting all three star conditions.

Additionally, E_i has the restriction that its first alphabet symbol produced may not be a_i — this is used to implement the third condition.

Y, Y', Y_i, Y'_i generate expressions whose prefix is an alphabet symbol. As a whole, these non-terminals correspond to Algorithm 2, and may be considered degenerate cases of Z and Z_i ; that is, trivial unions.

The tick-mark signifies that expressions of the form aa^* for $a \in \Sigma$ are disallowed, used to implement the second condition. The subscripted i signifies that the initial alphabet symbol may not be a_i , used to implement the third condition.

Z, Z_i generate non-trivial unions of expressions beginning with distinct alphabet symbols — Z corresponds to line 2 of Algorithm 1 and line 13 of Algorithm 2, while Z_i corresponds to line 5 of Algorithm 2.

The subscripted i signifies that none of initial alphabet symbols may be a_i , used to implement the third condition.

P_i, P'_i generate expressions beginning with the specified alphabet symbol a_i . They correspond to line 1 of Algorithm 2.

The tick-mark signifies that expressions may not have the prefix $a_i a_i^*$, used to implement the second condition.

Since the algorithm correctly returns a trie when run on the language represented by the trie, the correspondence between the algorithm and the grammar gives us the following result.

Theorem 7.5. *The grammar above is unambiguous and the generated regular expressions represent distinct regular languages.*

k	ordinary	reverse polish	alphabetic
1	$\Omega(1.3247^n)$	$\Omega(1.2720^n)$	$\Omega(2^n)$
2	$\Omega(2.7799^n)$	$\Omega(2.2140^n)$	$\Omega(7.4140^n)$
3	$\Omega(3.9582^n)$	$\Omega(2.8065^n)$	$\Omega(12.5367^n)$
4	$\Omega(5.0629^n)$	$\Omega(3.2860^n)$	$\Omega(17.6695^n)$
5	$\Omega(6.1319^n)$	$\Omega(3.6998^n)$	$\Omega(22.8082^n)$
6	$\Omega(7.1804^n)$	$\Omega(4.0693^n)$	$\Omega(27.9500^n)$

Table 4. Improved lower bounds for $R_k(n)$ with respect to size measure and alphabet size.

Table 4 lists the improved lower bounds for $R_k(n)$. These lower bounds were obtained via singularity analysis, as explained in Section 6, boot-strapping off the bounds in Table 2.¹

8 Upper bounds on enumeration of regular languages by regular expressions

Turning our attention back to upper bounds for $R_k(n)$, we develop grammars for regular expressions such that every regular language is represented by at least one shortest regular expression generated by the grammar, where a regular expression r of size n is said to be shortest if there is no expression r' of size less than n with $L(r) = L(r')$.

To this end, we consider certain “normal forms” for regular expressions, with the property that transforming a regular expression into normal form never increases its size. Again, size may refer to one of the various measures introduced before. With such a normal form, it suffices to enumerate all regular expressions in normal form to obtain improved upper bounds on $R_k(n)$ for various measures.

8.1 A grammar based on normalized regular expressions

We begin with a simple approach, which will be further refined later on. As concatenation and sum are associative, we consider them to be variadic operators taking at least 2 arguments and impose the condition that in any parse tree, neither of them are permitted to have themselves as children. Also, by the commutativity of the sum operator, we impose the condition that the summands of each sum appear in the following order: First come all summands which are terminal symbols, then all summands which are concatenations, and finally all starred summands. Also, we can safely omit all subexpressions of the form s^{**} , $s^* + \varepsilon$, $(s + \varepsilon)^*$, $s + \varepsilon + \varepsilon$: occurrences of these can be replaced with occurrences of s^* , s^* , s^* , and $s + \varepsilon$, respectively. Here the latter subexpressions have size no larger

¹The Maple worksheets used to derive these bounds can be accessed at the second author’s personal homepage via <http://math.stanford.edu/~jlee/automata/>

$S \rightarrow$	$Q \mid A \mid T \mid C \mid K$
$Q \rightarrow$	$A + \varepsilon \mid T + \varepsilon \mid C + \varepsilon$
$A \rightarrow$	$T + A_T \mid C + A_C \mid K + A_K$
$A_T \rightarrow$	$T \mid T + A_T \mid A_C$
$A_C \rightarrow$	$C \mid C + A_C \mid A_K$
$A_K \rightarrow$	$K \mid K + A_K$
$T \rightarrow$	$a_1 \mid a_2 \mid \cdots \mid a_k$
$C \rightarrow$	$C_0 C_0 \mid C_0 C$
$C_0 \rightarrow$	$(Q) \mid (A) \mid T \mid K$
$K \rightarrow$	$(A)^* \mid T^* \mid (C)^*$

Table 5. A simple unambiguous grammar for generating at least one shortest regular expression for each regular language.

$C \rightarrow$	$(Q)C_Q \mid (A)C_A \mid TC_T \mid KC_K$
$C_Q \rightarrow$	$(Q) \mid (Q)C_Q \mid C_A$
$C_A \rightarrow$	$A \mid (A)C_A \mid C_T$
$C_T \rightarrow$	$T \mid TC_T \mid C_K$
$C_K \rightarrow$	$K \mid KC_K$

Table 6. Rules for concatenation over unary alphabets, which in that case is commutative.

than the former ones, and this holds for all size measures considered. These observations immediately lend themselves for a simple unambiguous grammar, such as the one listed in Table 5. The meaning of the variables is as follows:

- S generates all regular expressions obeying the abovementioned format. Among them,
- Q generates those expressions of the form $r + \varepsilon$,
- A generates those of the form $r + s$, i.e. “additions”,
- T generates those which are terminal symbols,
- C generates those of the form rs , i.e. concatenations,
- C_0 generates the “factors” appearing inside concatenations (which are themselves not concatenations), and
- K generates those of the form r^* , i.e. Kleene stars;

finally, the “summands” in expressions of type A are subdivided into subtypes A_T , A_C and A_K , used for handling summands which are terminal symbols, concatenations, or Kleene stars, respectively.

In the special case of unary alphabets, not only union, but also concatenation (again viewed as a variadic operator) is commutative. In this case, we may impose a similar ordering of factors as done for summands, and thus we can replace the rule with C as left-hand side with the rules given in Table 6.

8.2 A grammar based on strong star normal form

We now refine the above approach by considering only regular expressions in strong star normal form [15], a notion that we recall in the following.

Since \emptyset is only needed to denote the empty set, and the need for ε can be substituted by the operator $L^? = L \cup \{\varepsilon\}$, an alternative syntax introduces also the $^?$ -operator and instead forbids the use of \emptyset and ε inside non-atomic expressions. The definition of strong star normal form is most conveniently given for this alternative syntax.

Definition 8.1. The operators \circ and \bullet are defined on regular expressions. The first operator is given by: $a^\circ = a$, for $a \in \Sigma$; $(r + s)^\circ = r^\circ + s^\circ$; $r^{?^\circ} = r^\circ$; $r^{*\circ} = r^\circ$; finally, $(rs)^\circ = rs$, if $\varepsilon \notin L(rs)$ and $r^\circ + s^\circ$ otherwise. The second operator is given by: $a^\bullet = a$, for $a \in \Sigma$; $(r + s)^\bullet = r^\bullet + s^\bullet$; $(rs)^\bullet = r^\bullet s^\bullet$; $r^{*\bullet} = r^{\bullet\circ*}$; finally, $r^{?^\bullet} = r^\bullet$, if $\varepsilon \in L(r)$ and $r^{?^\bullet} = r^{\bullet?}$ otherwise. The *strong star normal form* of an expression r is then defined as r^\bullet .

An easy induction shows that the transformation into strong star normal form preserves the described language, and that it is weakly monotone with respect to all usual size measures. We sketch a proof for the case of ordinary length.

Lemma 8.1. *Let r be a regular expression without occurrences of the symbol \emptyset , and let r^\bullet be its strong star normal form. Then $\text{ord}(r^\bullet) \leq \text{ord}(r)$.*

Proof Sketch. First of all, we may safely assume that r does not contain any subexpressions ruled out by the grammar of the previous section, such as $\varepsilon + \varepsilon$; the transformation into strong star normal form subsumes these reductions anyway.

Recall the definition of the auxiliary operator $^\circ$ in the definition of strong star normal form (Definition 8.1). The proof relies on the following claim: If $\varepsilon \in L(r)$ and $L(r) \neq \{\varepsilon\}$, then $\text{ord}(r^\circ) \leq \text{ord}(r) - 1$; otherwise, $\text{ord}(r^\circ) \leq \text{ord}(r)$. This claim can be proved by induction while excluding the cases $L(r) = \emptyset, \{\varepsilon\}$. The base cases are easy; the induction step is most interesting in the case $r = st$. If $\varepsilon \notin L(st)$, then $r^\circ = st$ and the claim holds; otherwise $r^\circ = s^\circ + t^\circ$ with $\varepsilon \in L(s)$ and $\varepsilon \in L(t)$. We can apply the induction hypothesis twice to deduce $\text{ord}(s^\circ) + \text{ord}(t^\circ) \leq \text{ord}(s) + \text{ord}(t) - 2$, and thus $\text{ord}(s^\circ + t^\circ) \leq \text{ord}(st) - 1$, as desired. Notice that, as union has lower precedence than concatenation, this step never introduces new parentheses. The induction step in the other cases is even easier. \square

Since every regular language is represented by at least one shortest regular expression in strong normal form (with respect to all three considered size measures), it suffices to enumerate those expressions in normal form. Our improved grammar will be based on the following simple observation on expressions in strong star normal form:

Lemma 8.2. *If s^* or $s + \varepsilon$ appears as a subexpression of an expression in star normal form, then $\varepsilon \notin L(s)$.* \square

To exploit this fact, for each subexpression we need to keep track of whether it denotes the empty word. This can of course be done with dynamic programming, by using rules

$S \rightarrow S^+ \mid S^-$	
$S^+ \rightarrow Q^+ \mid A^+ \mid C^+ \mid K^+$	$S^- \rightarrow A^- \mid T^- \mid C^-$
$Q^+ \rightarrow A^- + \varepsilon \mid T^- + \varepsilon \mid C^- + \varepsilon$	
$A^+ \rightarrow T^- + A_C^+ \mid C^- + A_C^+ \mid$ $A^- + A_C^+ \mid C^+ + A_C^+ \mid$ $K^+ + A_K^+$	$A^- \rightarrow T^- + A_T^- \mid C^- + A_C^-$ $A_T^- \rightarrow T^- \mid T^- + A_T^- \mid A_C^-$ $A_C^- \rightarrow C^- \mid C^- + A_C^-$
$A_C^+ \rightarrow C^+ \mid C^+ + A_C^+ \mid A_K^+$	
$A_K^+ \rightarrow K^+ \mid K^+ + A_K^+$	
$T^- \rightarrow a_1 \mid a_2 \mid \dots \mid a_k$	
$C^+ \rightarrow C_0^+ C_0^+ \mid C_0^+ C^+$	$C^- \rightarrow C_0^- C_0^- \mid C_0^- C_0^+ \mid C_0^+ C_0^- \mid$ $C_0^- C^- \mid C_0^- C^+ \mid C_0^+ C^-$
$C_0^+ \rightarrow (Q^+) \mid (A^+) \mid K^+$	$C_0^- \rightarrow (A^-) \mid T^-$
$K^+ \rightarrow (A^-)* \mid T^-* \mid (C^-)*$	

Table 7. A better unambiguous grammar generating at least one shortest regular expression (in strong star normal form) for each regular language.

such as $\varepsilon \in L(rs)$ iff $\varepsilon \in L(r)$ and $\varepsilon \in L(s)$. Since in addition every subexpression either denotes the empty word or not, it is easy to extend the above grammar to incorporate these rules while retaining the property of being unambiguous.

Notice that most variables now come in an ε -flavor (for example, the variable A^+) and in an ε -free flavor (for example, the variable A^-). Moreover, the summands inside sums appear in the following order, which is a refinement of the summand ordering devised previously: First come all summands which are terminal symbols, then all summands which are ε -free concatenations, then all concatenations with ε in the denoted language, and finally all starred summands. To illustrate this ordering, we give the most important steps of the unique derivation for the expression $a_1 + a_2a_3 + (a_4 + \varepsilon)(a_5 + \varepsilon) + a_6^*$:

$$\begin{aligned}
S &\Rightarrow^* A^- + A_C^+ \Rightarrow T^- + A_T^- + A_C^+ \Rightarrow a_1 + A_T^- + A_C^+ \\
&\Rightarrow a_1 + A_C^- + A_C^+ \Rightarrow a_1 + C^- + A_C^+ \Rightarrow^* a_1 + a_2a_3 + A_C^+ \\
&\Rightarrow a_1 + a_2a_3 + C^+ + A_C^+ \Rightarrow^* a_1 + a_2a_3 + (a_4 + \varepsilon)(a_5 + \varepsilon) + A_C^+ \\
&\Rightarrow a_1 + a_2a_3 + (a_4 + \varepsilon)(a_5 + \varepsilon) + A_K^+ \Rightarrow a_1 + a_2a_3 + (a_4 + \varepsilon)(a_5 + \varepsilon) + K^+ \\
&\Rightarrow^* a_1 + a_2a_3 + (a_4 + \varepsilon)(a_5 + \varepsilon) + a_6^*
\end{aligned}$$

The following proposition, giving the correctness of the improved grammar, can be proved by induction on the minimum required regular expression size. Table 8 lists the upper bounds obtained through this grammar.²

²The Maple worksheets used to derive these bounds can be accessed at the second author's personal homepage via <http://math.stanford.edu/~jlee/automata/>

Proposition 8.3. *The grammar in Table 7 is unambiguous and, for each regular language, generates at least one regular expression of minimal ordinary length (respectively: reverse polish length, alphabetic width) representing it.* \square

k	ordinary	reverse polish	alphabetic
1	$O(2.5946^n)$	$O(2.7422^n)$	} $O(k^n \cdot 21.5908^n)$
2	$O(4.2877^n)$	$O(3.9870^n)$	
3	$O(5.4659^n)$	$O(4.7229^n)$	
4	$O(6.5918^n)$	$O(5.3384^n)$	
5	$O(7.6870^n)$	$O(5.8780^n)$	
6	$O(8.7624^n)$	$O(6.3643^n)$	

Table 8. Summary of upper bounds on $R_k(n)$ for $k = 1, 2, \dots, 6$ and various size measures. For ordinary length, we used the simple grammar in Table 5, because the computation for the improved grammar ran out of computational resources. For reverse polish length, we used the simple grammar for bootstrapping the bounds.

k	ordinary	reverse polish	alphabetic
1	$O(2.1793^n)$	$O(2.0795^n)$	$O(10.9822^n)$
2	$O(3.8145^n)$	$O(3.3494^n)$	} $O(k^n \cdot 12.2253^n)$
3	$O(4.9019^n)$	$O(4.0315^n)$	
4	$O(5.8234^n)$	$O(4.6121^n)$	
5	$O(6.8933^n)$	$O(5.1268^n)$	
6	$O(7.9492^n)$	$O(5.5939^n)$	

Table 9. Summary of upper bounds for $k = 1, 2, \dots, 6$ and various size measures in the case of finite languages. For reverse polish length, we bootstrapped from the values in Table 8; for ordinary length, we bootstrapped the case $k = 2$ from the upper bound obtained for $k = 3$.

9 Exact enumerations

Tables 10 to 15 give exact numbers for the number of regular languages representable by a regular expression of size n , but not by any of size less than n .

We explain how these numbers were obtained.³ Using the upper bound grammars described previously, a dynamic programming approach was taken to produce (in order of increasing regular expression size) the regular expressions generated by each non-terminal. To account for duplicates, each regular expression was transformed into a DFA,

³The C++ source code of the software used to compute these numbers can be accessed at the second author's personal homepage via <http://math.stanford.edu/~jlee/automata/>

minimized and relabelled via a breadth-first search to produce a canonical representation. Using these representations as hashes, any regular expression matching a previous one generated by the same non-terminal was simply ignored.

k	1	2	3	4
1	3	4	5	6
2	1	4	9	16
3	2	11	33	74
4	3	28	117	336
5	3	63	391	1474
6	5	156	1350	6560
7	5	358	4546	28861
8	8	888	15753	128720
9	9	2194	55053	578033
10	14	5665	196185	2624460

Table 10. Ordinary length, finite languages

k	1	2	3	4
1	3	4	5	6
2	2	6	12	20
3	3	17	48	102
4	4	48	192	520
5	5	134	760	2628
6	9	397	3090	13482
7	12	1151	12442	68747
8	17	3442	51044	354500
9	25	10527	211812	1840433
10	33	32731	891228	

Table 11. Ordinary length, general case

k	1	2	3	4
1	3	4	5	6
3	2	7	15	26
5	3	25	85	202
7	5	109	589	1917
9	9	514	4512	20251
11	14	2641	37477	231152
13	24	14354	328718	2780936
15	41	81325	2998039	
17	71	475936		
19	118	2854145		

Table 12. Reverse polish length, finite languages

k	1	2	3	4
1	3	4	5	6
2	1	2	3	4
3	2	7	15	26
4	2	13	33	62
5	3	32	106	244
6	4	90	361	920
7	6	189	1012	3133
8	7	580	3859	13529
9	11	1347	11655	48388
10	15	3978	43431	208634

Table 13. Reverse polish length, general case

10 Conclusion and open problems

In this chapter, we discussed various approaches to enumerating regular expressions and the languages they represent, and we used algebraic and analytic tools to compute upper and lower bounds for these enumerations. Our upper and lower bounds are not always very close, so an obvious open problem (or class of open problems) is to improve these bounds. Other problems we did not examine here involve enumerating interesting subclasses of regular expressions. For example, in linear expressions, every alphabet symbol

k	1	2	3	4
0	2	2	2	2
1	2	4	6	8
2	4	24	60	112
3	8	182	806	2164
4	16	1652	13182	51008
5	32	16854	242070	1346924
6	64	186114	4785115	

Table 14. Alphabetic width, finite languages

k	1	2	3	4
0	2	2	2	2
1	3	6	9	12
2	6	56	150	288
3	14	612	3232	9312
4	30	7923	82614	357911
5	72	114554	2332374	
6	155	1768133		

Table 15. Alphabetic width, general case

occurs exactly once. In addition to the intrinsic interest, enumerating subclasses may provide a strategy for improving the lower bounds for the general case.

References

- [1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.
- [2] F. Bassino and C. Nicaud. Enumeration and random generation of accessible automata. *Theoret. Comput. Sci.* **381**(1–3) (2007), 86–104.
- [3] D. Callan. A determinant of Stirling cycle numbers counts unlabeled acyclic single-source automata. *Discrete Math. & Theoret. Comput. Sci.* **10** (2008), 77–86.
- [4] N. Chomsky and M. P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pp. 118–161. North Holland, Amsterdam, 1963.
- [5] D. A. Cox, J. Little, and D. O’Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer-Verlag, 3rd edition, 2007.
- [6] M. Domaratzki. Improved bounds on the number of automata accepting finite languages. *Internat. J. Found. Comp. Sci.* **15** (2004), 143–161.
- [7] M. Domaratzki. Combinatorial interpretation of a generalization of the Genocchi numbers. *J. Integer Sequences* **7** (2004), 04.3.6 (electronic).
- [8] M. Domaratzki. Enumeration of formal languages. *Bull. European Assoc. Theor. Comput. Sci.*, No. 89, (June 2006), 117–133.
- [9] M. Domaratzki, D. Kisman, and J. Shallit. On the number of distinct languages accepted by finite automata with n states. *J. Automata, Languages, and Combinatorics* **7**(4) (2002), 469–486.
- [10] A. Ehrenfeucht and P. Zeiger. Complexity measures for regular expressions. *J. Comput. System Sci.* **12** (1976), 134–146.
- [11] K. Ellul, B. Krawetz, J. Shallit, and M.-w. Wang. Regular expressions: new results and open problems. *J. Automata, Languages, and Combinatorics* **10**(4) (2005), 407–437.

- [12] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.
- [13] S. Ginsburg. *An Introduction to Mathematical Machine Theory*. Addison-Wesley, 1962.
- [14] I. P. Goulden and D. M. Jackson. *Combinatorial Enumeration*. Wiley, 1983.
- [15] H. Gruber and S. Gulan. Simplifying regular expressions. a quantitative perspective. In A.-H. Dediu, H. Fernau, and C. Martín-Vide, editors, *Proc. 4th Int'l Conf. on Language and Automata Theory and Applications (LATA)*, LNCS. Springer-Verlag, 2010. To appear.
- [16] F. Harary. The number of functional digraphs. *Math. Annalen* **138** (1959), 203–210.
- [17] F. Harary. Unsolved problems in the enumeration of graphs. *Magyar Tud. Akad. Math. Kutató Int. Közl.* **5** (1960), 63–95.
- [18] F. Harary. Combinatorial problems in graphical enumeration. In E. Beckenbach, editor, *Applied Combinatorial Mathematics*, pp. 185–217. Wiley, 1964.
- [19] F. Harary and E. Palmer. Enumeration of finite automata. *Inform. Control* **10** (1967), 499–508.
- [20] M. A. Harrison. A census of finite automata. In *Proc. 5th Annual Symposium on Switching Circuit Theory and Logical Design*, pp. 44–46. IEEE Press, 1964.
- [21] M. A. Harrison. A census of finite automata. *Canad. J. Math.* **17** (1965), 100–113.
- [22] R. Hartshorne. *Algebraic geometry*, Vol. 52 of *Graduate Texts in Mathematics*. Springer-Verlag, 1977.
- [23] Markus Holzer and Martin Kutrib. Scientific applications of language methods. Vol. 2 of *Mathematics, Computing, Language, and Life: Frontiers in Mathematical Linguistics and Language Theory*, chapter Descriptive Complexity — An Introductory Survey, pp. 1–58. World Scientific, 2010.
- [24] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
- [25] L. Ilie and S. Yu. Algorithms for computing small NFAs. In *Proc. 27th Symposium, Mathematical Foundations of Computer Science 2002*, Vol. 2420 of LNCS, pp. 328–340. Springer-Verlag, 2002.
- [26] D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 3rd edition edition, 1997.
- [27] A. D. Korshunov. On asymptotic estimates of the number of finite automata. *Diskretnyi Analiz*, No. 6, (1966), 35–50. In Russian.
- [28] A. D. Korshunov. Asymptotic estimates of the number of finite automata. *Kibernetika* **3**(2) (1967), 12–19. In Russian. English translation in *Cybernetics* **3** (2) (1967), 9–14.
- [29] A. D. Korshunov. A survey of certain trends in automata theory. *Diskretnyi Analiz*, No. 25, (1974), 19–55, 62. In Russian.
- [30] A. D. Korshunov. The number of automata and boundedly determined functions. Hereditary properties of automata. *Dokl. Akad. Nauk SSSR* **221** (1975), 1264–1267. In Russian. English translation in *Soviet Math. Doklady* **16** (1975), 515–518.
- [31] A. D. Korshunov. Enumeration of finite automata. *Problemy Kibernetiki*, No. 34, (1978), 5–82, 272. In Russian.
- [32] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Springer-Verlag, 1985.
- [33] J. Lee and J. Shallit. Enumerating regular expressions and their languages. In M. Domaratzki, A. Okhotin, K. Salomaa, and S. Yu, editors, *Proc. 9th Int'l Conf. on Implementation and Application of Automata (CIAA)*, Vol. 3317 of LNCS, pp. 2–22, 2005.

- [34] E. Leiss. Constructing a finite automaton for a given regular expression. *SIGACT News* **12**(3) (Fall 1980), 81–87.
- [35] V. A. Liskovets. The number of connected initial automata. *Kibernetika* **5**(3) (1969), 16–19. In Russian. English translation in *Cybernetics* **5** (1969), 259–262.
- [36] V. A. Liskovets. Exact enumeration of acyclic deterministic automata. *Disc. Appl. Math.* **154**(3) (2006), 537–551.
- [37] E. M. Livshits. Asymptotic formula for the number of classes of isomorphic autonomous automata with n states. *Ukrainskii Matematicheskii Zhurnal* **16** (1964), 245–246. In Russian.
- [38] J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, 3rd edition, 2003.
- [39] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IRE Trans. Electron. Comput.* **EC-9** (1960), 39–47.
- [40] C. Nicaud. Average state complexity of operations on unary automata. In M. Kutylowski, L. Pacholski, and T. Wierzbicki, editors, *Proc. 24th Symposium, Mathematical Foundations of Computer Science 1999*, Vol. 1672 of *LNCS*, pp. 231–240. Springer-Verlag, 1999.
- [41] A. Panholzer. Gröbner bases and the defining polynomial of a context-free grammar generating function. *J. Automata, Languages, and Combinatorics* **10** (2005), 79–97.
- [42] C. Pomerance, J. M. Robson, and J. Shallit. Automaticity II: Descriptive complexity in the unary case. *Theoret. Comput. Sci.* **180** (1997), 181–201.
- [43] C. E. Radke. Enumeration of strongly connected sequential machines. *Inform. Control* **8** (1965), 377–389.
- [44] D. Raymond and D. Wood. *Grail*: a C++ library for automata and expressions. *J. Symbolic Comput.* **17** (1994), 341–350.
- [45] R. C. Read. A note on the number of functional digraphs. *Math. Annalen* **143** (1961), 109–110.
- [46] R. W. Robinson. Counting strongly connected finite automata. In Y. Alavi, G. Chartrand, L. Lesniak, D. R. Lick, and C. E. Wall, editors, *Graph Theory with Applications to Algorithms and Computer Science*, pp. 671–685. Wiley, 1985.
- [47] W. Rudin. *Real and Complex Analysis*. McGraw-Hill, 1966.
- [48] J. Shallit and Y. Breitbart. Automaticity I: Properties of a measure of descriptive complexity. *J. Comput. System Sci.* **53** (1996), 10–25.
- [49] R. P. Stanley. *Enumerative Combinatorics*, Vol. 62 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1999.
- [50] V. A. Vyssotsky. A counting problem for finite automata. Technical report, Bell Telephone Laboratories, May 1959.
- [51] H. Wilf. *Generatingfunctionology*. A. K. Peters, 2006.
- [52] D. Ziadi. Regular expression for a language without empty word. *Theoret. Comput. Sci.* **163** (1996), 309–315.

Abstract. In this chapter we discuss the problem of enumerating distinct regular expressions by size and the regular languages they represent. We discuss various notions of the size of a regular expression that appear in the literature and their advantages and disadvantages. We consider a formal definition of regular expressions using a context-free grammar.

We then show how to enumerate strings generated by an unambiguous context-free grammar using the Chomsky-Schützenberger theorem. This theorem allows one to construct an algebraic equation whose power series expansion provides the enumeration. Classical tools from complex analysis, such as singularity analysis, can then be used to determine the asymptotic behavior of the enumeration.

We use these algebraic and analytic methods to obtain asymptotic estimates on the number of regular expressions of size n . A single regular language can often be described by several regular expressions, and we estimate the number of distinct languages denoted by regular expressions of size n . We also give asymptotic estimates for these quantities. For the first few values, we provide exact enumeration results.